As I work on resources, here are thoughts, issues that come up. These should morph into a real wiki page describing the resources for users!

Note that I have separate pages on Resource FV detectors and on how dynamic europa is doing things, all of which is relevant to this and should be included in the consolidation.

Issues

- LowerLevelMax and UpperLevelMin are weird cases. I'm tempted to remove them. However, for
- Related to that, there seems to be no built-in support for having flexible quantities. Is that true - ie is decision making always predicated on upper/lower bounds or are there heuristics based on this. My assumption is that a flaw will be resolved with an ordering decision, when in fact it could be resolved by choosing to constrain the quantity consumed, for example
- !OpenWorldFVDetector, in addition to reporting violations only when consumption/production couldn't appear out of thin air to fix the levels (based on max production/consumption limits) is (about to change?) reporting flaws based on LowerLevelMax? and UpperLevelMin?, which seems to be more like a 'grounded' approach. For example, suppose A is the only activity, uses [5,10] units, and there are 2 available. The closed world detector reports a flaw, whereas the open world detector does not! It seems like that could be what a user wants, but should probably be something they code themselves for the situation. ASK MIKE ABOUT THIS, THEN ELIMINATE THIS DIFFERENCE
- Related to that, notice that we can't simply ground quantities with the grounded profiles, since they
- Best approach. Create the generic GroundedInstant, a profile that calculates them, perhaps a single grounded fv detector, but leave weird special cases like the above (using LowerLevelMax? etc) to user-defined project-specific code.
- Note that flow profiles ONLY use the lower/upper levels (ie 2 of the 4 profiles) and completely ignore instantaneous/cumulative production/consumption!

FLAW: Possible problem - you could set times/quantities within current bounds and cause trouble, but there are (probably) ways to avoid it (ordering decisions, etc) VIOLATION: Impossible problem - ie you must undo existing decisions to have any hope!

TODO: Lengthen these notes so I'll understand them next year, not just next month!

How things work

!ThreatDecisionPoint::handleInitialize has the following steps:

1. getOrderingChoices (a bunch of pairs of tokens for which a precedence constraint would probably help)
2. create filter (s?) based on the xml config file
3. Sort the ordering choices based on xml config file

Then, handleExecute looks at next choice in the sorted list and imposes a precedence constraint.

Grounded Profile

Proposed semantics:

- Violations based on non-grounded values, and can use Timetable profiles (or anything fast and loose)

- Flaws based on grounded temporal (but not quantity) variables

Profiles:

- all instantaneous/cumulative production/consumption can be handled by treating mins as ungrounded (for violations) and maxes as grounded (for flaws)
- Because lower/upper profiles both used for violations, need upper/lower grounded profiles (both upper and lower because quantities can be variable) in addition, so subclass Instant (ugly alternative
- getTransactionsToOrder should return inst->getGroundedTransactions

Instant:

- Two extra profiles, and an extra set containing the grounded transactions at a given instant (bit tricky?) (ie getGroundedTransactions method)
- DOH: A flawed instant could easily have a single grounded transaction, so all overlapping ones need to be included BUT not ones that are just around because of violations SOS...
    - ♦ SOS: I don't really understand this yet!
- Probably need Instant::getGroundedTransactions so that ThreatHandler? only considers reording pairs that actually matter!

FV detectors:

- Need to check the instantaneous/cumulative production/consumption per the above assumption.
- Need to check the above grounded profiles specially, so probably want versions of both open/closed with the new methods

Easy alternative (suggested by Paul):

- Create a grounded profile that ignores violations (ie just use lower/upper bounds as the grounded profiles since I don't need anything else)
- Note that this would allow for my failed generalization that adjusts where variables are added to instants, instead of doing adjustments at a lower level.
- Therefore, this would probably also let ThreatManager? etc work out of the box. For example, even though a flawed instant might only have single transaction associated with it (even though others overlap), the code getOrderingChoices call to the resource class will get all tokens that overlap (sweet)

TODO:

- How to organize the fleet of FV detectors
- Since the open-world case seems so unlikely in practice, (ie really want a NoViolationsFVDetector), shouldn't that be left to an individulal user?

Unary Resource Profile

- How to implement. Can we copy what Timeline does, or implement something even faster?